On-the-fly simplification of genetic programming models

Noman Javed
London School of Economics and Political Science
n.javed3@lse.ac.uk

Fernand Gobet
London School of Economics and Political Science
f.gobet@lse.ac.uk

ABSTRACT

The last decade has seen amazing performance improvements in deep learning. However, the black-box nature of this approach makes it difficult to provide explanations of the generated models. In some fields such as psychology and neuroscience, this limitation in explainability and interpretability is an important issue. Approaches such as genetic programming are well positioned to take the lead in these fields because of their inherent white box nature. Genetic programming, inspired by Darwinian theory of evolution, is a population-based search technique capable of exploring a high-dimensional search space intelligently and discovering multiple solutions. However, it is prone to generate very large solutions, a phenomenon often called "bloat". The bloated solutions are not easily understandable. In this paper, we propose two techniques for simplifying the generated models. Both techniques are tested by generating models for a well-known psychology experiment. The validity of these techniques is further tested by applying them to a symbolic regression problem. Several population dynamics are studied to make sure that these techniques are not compromising diversity – an important measure for finding better solutions. The results indicate that the two techniques can be both applied independently and simultaneously and that they are capable of finding solutions at par with those generated by the standard GP algorithm – but with significantly reduced program size. There was no loss in diversity nor reduction in overall fitness. In fact, in some experiments, the two techniques even improved fitness.

CCS CONCEPTS

• **Computing methodologies → Genetic programming**;

KEYWORDS

Evolutionary Computing, Genetic Programming, Simplification

ACM Reference Format:
Noman Javed and Fernand Gobet. 2021. On-the-fly simplification of genetic programming models. In *Proceedings of ACM SAC Conference (SAC'21).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3412841.3441926

1 INTRODUCTION

Deep learning, having theoretical foundations in neural networks, has dominated the arena of machine learning during the last decade. The primary reason behind this dominance has been its impressive performance. Factors such as the increase in computational power and the availability of software resources have also contributed to the adoption of deep learning in almost every field. Despite impressive performance and widespread acceptability, there exist communities in psychology, neuroscience, and many other domains that are reluctant to accept the solutions proposed by deep learning. The main cause of their concern is the black-box nature of deep learning, which conceals the relationship between inputs and outputs. These communities want to understand the process of transformation of input to output. Understanding this relationship is of utmost importance for interpreting and explaining the models. That is why explainable artificial intelligence is gaining traction and is subject of much research [2, 7, 9, 22].

We can classify research in explainable artificial intelligence into two categories. One way is to attach explanations to the black-box models [8]. Other approaches are inherently explainable because of their white-box nature. One such technique is genetic programming (GP). It is not only intrinsically transparent but can also be employed to generate explanations of black-box models [8, 10]. Thus, it has the potential to achieve good performance without compromising explainability.

Despite the inherent advantage of its white-box nature, GP is not as widely used as deep learning. One of the limiting factors is the complexity of the solutions generated by GP, where complexity refers to the size of the individual, and the size means the number of nodes in a tree. The programs generated by GP tend to grow in size without significantly improving performance. This phenomenon is called "bloat". This problem has been identified from the very early days of GP [3] and the literature is replete with methods for bloat identification and control [13, 21]. One of the well-know bloat control techniques is simplification [12]. The logic behind simplification is to reduce the size of the individual. It does so by identifying the redundant parts that are not contributing to the improvement in fitness. It then removes these redundant non-contributing parts, resulting in a smaller sized individual with the same or better fitness.

In this paper, we propose two simplification techniques and report their results in two tasks, comparing them with the results of standard GP. First, we apply them to the task of generating explanatory models for a well-known psychology experiment known as the Posner cueing task, the aim of which is to understand attentional mechanisms in human cognition. In this experiment, a cue first appears (e.g. an arrow pointing left or right). Then, participants have to respond as rapidly as possible to a target presented to the left or right of a fixation point or the left or right ear. To further verify the performance and applicability of these simplification techniques, we also apply them to a symbolic regression problem. GP has already been used extensively for performing symbolic regression. We found that both these techniques generate solutions of significantly reduced size in both the Posner cueing and symbolic regression problems. It was further observed that this reduction in size does not come at a price of fitness or execution time.

SAC'21, March 22-March 26, 2021, Gwangju, South Korea
2021. ACM ISBN 978-1-4503-8104-8/21/03…$15.00
https://doi.org/10.1145/3412841.3441926

2 SIMPLIFICATION

Since GP is primarily fitness driven, it tends to prefer fitter individuals by assigning them a high probability of selection to breed the next generation of individuals. This approach has several side effects.

- The size of individuals keeps on increasing over generations. At the same time, the rate of improvement in fitness decreases gradually. So, a little improvement in fitness comes at the cost of a significant increase in size.
- Multiple copies of an individual can exist in a generation and across generations.
- Redundant genetic material. Many individuals share the same copies of genetic materials in the form of common subtrees.

Although a limited amount of redundancy is not harmful, an increase in redundancy means a decrease in diversity, which in turn means a meager exploration of the search space; this may result in premature convergence.

To address these side effects, researchers propose several simplification techniques in the literature. The core idea behind all of them is to identify and get rid of the redundant and non-contributing parts, also called *introns*. Simplification algorithms mainly differ in the way they spot similarities. Some of them use syntactic similarity to classify a part of an individual as redundant, whereas others use semantic similarity to achieve the same aim.

The earliest and most frequently used form of bloat control is to place a constraint on the size of the program. The target of this constraint is usually the depth of the GP trees. The limitation of this approach is that it cannot find solutions lying beyond these limits. Moreover, the individuals within those limits can still contain redundant and unwanted code. Another frequently used idea is to make program size an integral part of the fitness function. In this way, the individuals bigger in size get penalised, thus making it less likely for them to take part in reproduction. These sorts of approaches are categorised under parsimony pressure.

Unlike the above mentioned indirect ways of controlling bloat, algebraic simplification [23, 24] is inspired by the principles of algebra. It replaces a part of code with an algebraic equivalent that is smaller in size. In this way, it guarantees that the fitness of the individual will remain unaffected by the process of simplification. The limitation of algebraic simplification is that domain experts need to develop equivalence rules. This process may be simple for mathematical problems but can pose significant challenges for other types of problems. Probably because of this reason, it has been mostly applied to symbolic regression and classification problems. Another limitation is that it never considers the fact that two different-looking individuals can yield the same behaviour; thus, they are semantically equivalent. The algebraic approach is unable to simplify such cases.

Numerical simplification techniques work at the semantic level rather than the algebraic level [11]. They work out by calculating the contribution of a node in its parent's fitness. If the contribution is significant, the child node replaces its parent node. Similarly, constants can replace some nodes. The first limitation of this approach is that the process is localised and limited to just one node. Secondly, thresholds must be defined very carefully and can have a significant impact. Another issue is the maintenance of minimum and maximum values to take pruning decisions. These limitations are addressed by defining a mechanism of accepting or rejecting pruning proposals based on permutation tests [18]. In another variant [14], the authors evaluate subtrees against some predefined regression points. Based on this evaluation, these subtrees are replaced by simpler subtrees. The main limitation of this approach is the penalty incurred in the form of computational cost. The authors of [4] have used a very similar technique of replacing a subtree with a terminal. They further extended their approach and proposed to replace a subtree with another subtree with approximately the same semantics [5, 15]. In [6], the authors proposed five different simplification techniques inspired from epigenetics. Instead of completely removing a gene, they silence it to retain the chance that mutation may unsilence it at some point in time. They used PushGP as the platform for implementing these simplification algorithms.

A relatively new approach of bloat control is using the execution time of individuals rather than their size [6, 19]. Our proposed techniques can be categorised under numerical simplification as they are based on fitness rather than syntactic similarity. It is currently based on program size but can be easily extended to time-driven simplification. Another major difference is that both our simplification algorithms work as part of an evolutionary run rather than post-processing of the individuals.

3 PROPOSED SIMPLIFICATION TECHNIQUES

3.1 Generationwide Simplification (Gws)

Since GP generates individuals through crossover, they contain the genetic material of both the parents. Some of this genetic material is important and contributing to the better fitness of the individual, while some of it can be of no use. Thus, there is a possibility of reducing the size of an individual by keeping a copy of good genetic material and removing the one that is not contributing. The idea is to replace a less fit parent individual with a child of better fitness. Since a child is a subtree of a parent individual, its size will be smaller than the parent tree.

We apply this simplification mechanism to all individuals of every kth generation. At every kth generation, instead of breeding the new individuals, using crossover or mutation, all individuals are subject to the simplification procedure. Every individual produces several child subtrees, whose fitness is re-computed. A hash value of every subtree is computed and is stored. In this way, multiple instances of a subtree – whether they exist in an individual or within different individuals – are stored only once. This process will create a new population of individuals that may or may not exceed the specified population size. If the number of newly generated individuals is greater than the population size, some of them having low fitness values will be discarded. But in the other case, the remaining individuals are generated following the routine process of crossover and mutation.

This idea has the following potential merits:

- Getting rid of the multiple copies of individuals
- Possibility of unlocking good genetic material locked within an individual. This situation happens when a subtree has better fitness than the parent. But because of the other genetic material in the parent individual, it cannot express itself.

- Getting rid of low-quality genetic material.
- No growth in terms of the size of the individuals. In the worst case – when every individual is of higher fitness than all of their children – they will remain untouched and become a part of the population. However, in the average case, there will be a reduction in the size of the individuals.
- No extra computational cost. We compute a hash value of every individual and store its fitness against that. So, redundant copies of individuals have no impact as they will undergo fitness computation only once.

3.2 Pruning as an Operator

In contrast to the generationwide simplification, pruning operates at the individual level rather than at the population level. The motivation behind this is to prune only those individuals who have a high probability of being selected as parents. These are the individuals of relatively higher fitness than the rest of the population. Thus, we applied pruning as an operator in every generation after calculating the fitness and sorting the individuals by fitness. Pruning selects the top few percent individuals of a population, based on fitness, and prunes them. This percentage is called the pruning rate, which can be varied. The individual selected for pruning undergoes a lossless pruning process. The pruning operator generates a set of subtrees of an individual and computes their fitness. The fittest subtree replaces the parent tree. If the fitness of the parent tree is best, it can retain its place without being impacted by the pruning operator.

Pruning, as an operator, offers the following benefits:

- Reduction in size of relatively fit individuals
- It does not interfere with the normal working of other genetic operators
- Very low computational overhead. This overhead depends on the pruning rate. With high pruning rates, it can be costly.

You can notice the similarity of the simplification procedure between both these techniques. They mainly differ in their application, where the former works at the generational level and the latter operates at the individual level. There are some other subtle differences. For example, in pruning, after splitting, the replacement of an individual comes from one of its children. Whereas in generationwide simplification, an individual and all its children may disappear from the population because of low fitness values; hence the one-to-one replacement of parent and child is not guaranteed.

4 EXPERIMENTS AND RESULTS

To test our proposed simplification algorithms, we use two problems of a completely different nature. The first is a well-known psychology experiment to understand the attentional mechanisms in human cognition, and the task is to find a model accounting for the human data. The other one is a classical symbolic regression problem.

4.1 Posner's cueing experiment

The experiment we used was carried out by Arjona et al.[1] and is a variation of Posner's cueing task [16]. The general aim of this task is to understand the role of attention on rapid perceptual decision making. In Arjona et al.'s experiment, participants are seated in front of a computer display and fixate a white cross at the centre of the screen. The cross is then replaced by an arrow, pointing either to the left or the right (visual cue). This is followed by the presentation of a sound in either the left or right ear (auditory stimulus). Participants are asked to press the button corresponding to the side of the auditory stimulus, using the index finger of the left or right hand. They have one second to do so. After a short pause, this sequence of events is repeated in the following trial.

The detailed time course of events is as follows:

(1) The central white cross is fixated for 300 milliseconds (ms)
(2) The visual arrow is displayed for 300 ms
(3) The central white cross is shown for 370 ms
(4) The auditory stimulus is presented for 100 ms
(5) The white cross is shown while the participant provides a response (within 1,000 ms)

While the auditory stimuli were randomly presented to the left or right ear with equal probability, the cue validity was systematically manipulated. In the 50% validity condition, the visual cue was valid (i.e. correctly predicted the auditory stimulus) in 50% of the trials. In the 68% validity condition, the cue was predictive in 68% of the trials. Finally, in the 86% validity condition, the cue was predictive in 86% of the trials. The trials were organised in 6 blocks of 100 trials, with 2 blocks for each cue validity condition. Thirty participants took part in Arjona et al.'s (2016) experiment.

Arjona et al. collected both behavioural data and electrophysiological data. In our simulations, we used only the behavioural data (response time and the number of errors). The main results were as follows. There was a significant cueing effect, as the response times were faster in the valid trials than in the invalid trials; the effect increased together with the proportion of valid trials (86% > 68% > 50%). Accuracy was also affected by cue validity: as the percentage of cue validity increased, the percentage of incorrect responses increased in the invalid trials, compared to the valid trials.

The simulations implemented the key features of the human experiment described above, thus producing 12 data points (3 cue validities (50%, 68% and 86%) × 2 types of trials (valid and invalid) × 2 dependent variables (response time and the number of errors)). The main simplification in the simulations was that the stimuli were presented symbolically and not as bitmaps and physical sounds. GP constructed models by combining terminals and operators, which all had a simulated time cost, the value of which was based on the psychological literature. The operators implementing learning employed Rescorla and Wagner's rule [17]:

$$\Delta V = \alpha(\lambda - V)$$

where ΔV is the change in the strength of association between cue and stimulus, α is the rate of change, λ the maximal value of the strength of association, and V the current strength of association. The terminals and operators used in the simulations were (a) terminals for inputting the cue and the stimuli, and responding; (b) operators for inputting and retrieving information in short-term memory (STM), and for comparing two elements in STM; (d) operators for carrying sequences of actions; (e)"waiting" operators, which did not do anything except increase a model's clock; (f) operators for directing attention; (g) operators for matching cue-stimulus probability or learning it using Rescorla and Wagner rule; and (h) IF and NIL. Table 1 presents the detail of the operators used.

Operator	Arity	Description
access-stm-1 (2 or 3)	0	Put item in STM slot 1 (2 or 3) into current value
compare-1-2 (1-3 or 2-3)	0	Current value is true/false if STM item 1/2/3 = item 2/3/1
if	3	Selects between two operators based on current value
prog2 (3 or 4)	2	Sequentially do two (three or four) operators
put-stm	0	Push current value on to STM
wait-200 (1000 or 1500)	0	Waits for 200 ms (1,000 or 1,500 ms)
nil	0	Set current value to 0 ('false')
respond-cue	0	Uses only the cue to predict the stimulus
respond-stimulus	0	Always responds with the stimulus
match-probability	0	Uses the predictive validity of the cue, which is assumed to be known
ResWagner-update	0	Learns by updating the predictive validity of the cue
ResWagner-cue-stimulus	0	Uses the cue and the strength of association to predict the stimulus
ResWagner-cue-priming-stimulus	0	Uses both the cue and the perception of the stimulus to select the stimulus

Table 1: Operators for modelling Posner's Task

4.1.1 Experimental Setup. We used the GP library proposed by Koza [12] to implement the two tasks and the simplification algorithms. Steel bank Common Lisp (SBCL) [20] version 1.5.5 was used to run the experiments. We ran the experiment 10 times, where each run comprised 100 generations. The population size of each generation was 1,000. Generationwide simplification was applied after every 25th generation while the pruning rate was 2 percent.

4.1.2 Results. The top ten individuals of every run, in terms of fitness, were compared using their average and maximum program sizes (see figure 1). The maximum reduction in program size occurred by applying both simplification schemes. However, for the large-sized individuals, generationwide simplification was better in limiting their growth. You can notice the small boxes of simplification schemes as compared to the plain GP, indicating that these schemes were successful in reducing the sizes of most of the individuals.

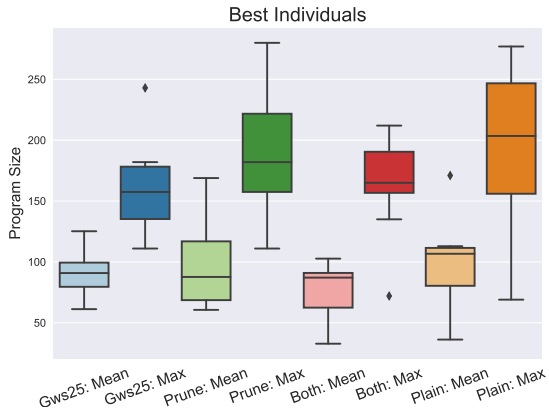

Figure 1: Posner Task - Top 10 / Run: Program Size

We compared these top 10 individuals of every run in terms of fitness. Here in figure 2, we present the mean and minimum fitness. A minimum is better because it reflects the error with respect to the original. You can notice that, in terms of minimum fitness, plain GP is best. However, other schemes are not very far from it. The smaller areas of simplification schemes suggest that most of the individuals are very near to each other in terms of fitness.

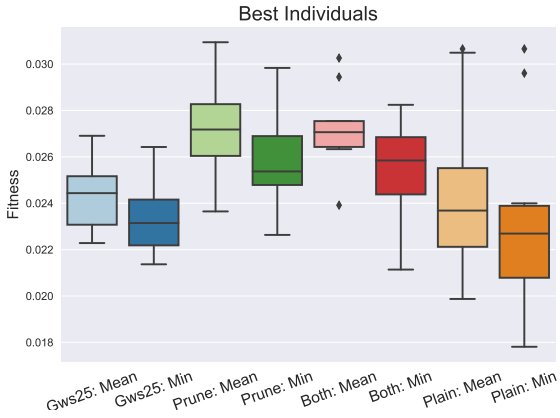

Figure 2: Posner Task - Top 10 / Run: Fitness

Generation specific statistics demonstrate the performance of simplification schemes as a part of the evolutionary journey. Figure 3 shows the comparison of the size of the fittest 20 individuals of every generation. It is evident from the figure that both the simplification schemes perform better than the plain GP. In some cases, the difference is very significant. This reduction in program size does not come at the cost of diversity, as is evident from the bottom-most graph of the same figure. This graph shows the plot of the standard deviation of the program size.

At what cost this reduction in size happened is a pertinent question at this stage. Two relevant cost measures are a cost in terms of fitness and a cost in terms of loss in diversity. To measure the first cost, we compared the fitness of the fittest 20 individuals of all generations (see figure 4). One can observe that the plain line is slightly lower than the other lines most of the time, thus indicating better fitness when no simplification is applied. However, the

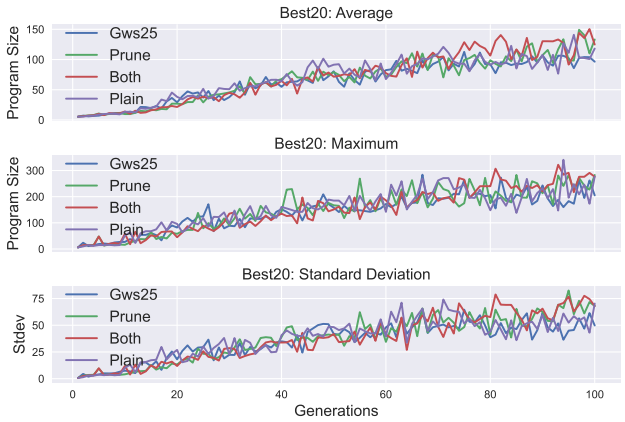

Figure 3: Posner Task - Best 20: Program Size

four lines are very near, hence pointing out the negligible cost of simplification in terms of fitness.

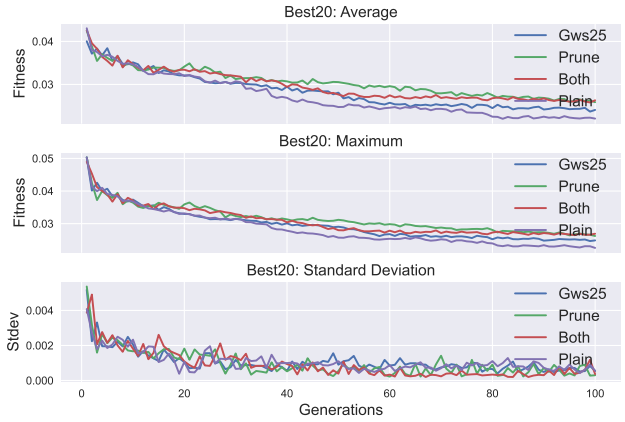

Figure 4: Posner Task - Best 20: Fitness

So far, the story is about the fittest individuals. To measure how much simplification is impacting the overall dynamics of the evolutionary progress, we compare the program sizes, fitness, and execution time taken by individuals. The scale of the comparison is at the population level to get a population-level view. Figure 5 compares the program sizes in terms of average program size and maximum program size. It is evident from the graphs that simplification algorithms perform better than the plain GP. These simplification algorithms are successful not only in reducing the size of the better individuals but also have a positive impact on the whole population.

To verify the cost paid in terms of fitness and loss in diversity, we compared population-wide fitness. The comparison is shown in figure 6. We observe no significant loss in terms of fitness. A comparison of diversity, both in terms of fitness and program size, was captured by calculating the standard deviation as presented in figure 7. Again, no loss in diversity is evident from the graph.

Figure 5: Posner Task - Comparison of Program Sizes

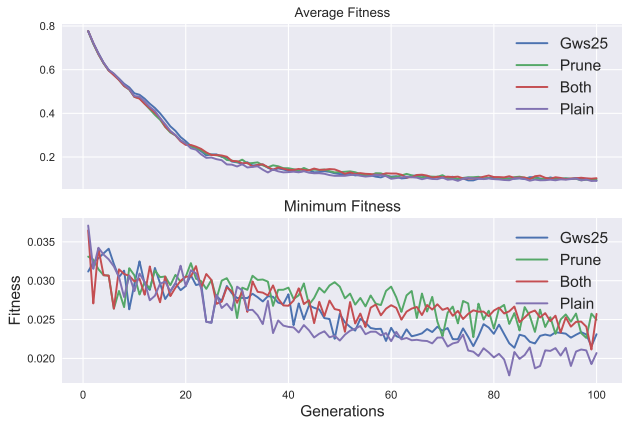

Figure 6: Posner Task - Comparison of Fitness

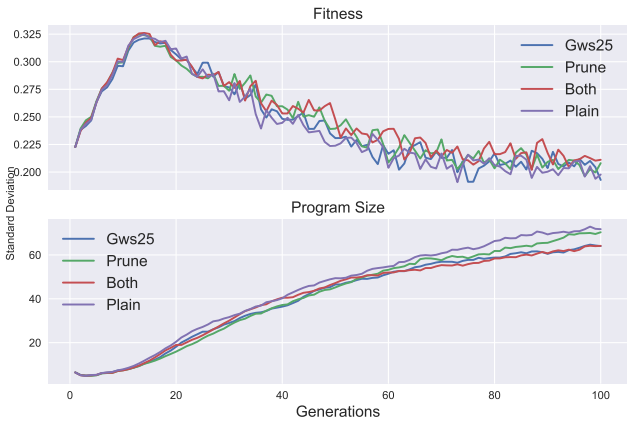

Figure 7: Posner Task - Comparison of Diversity

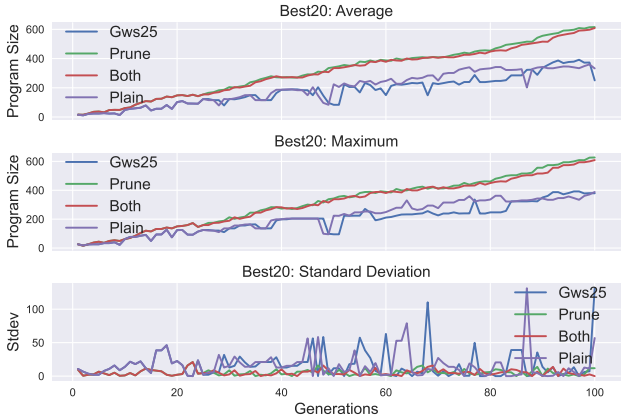

Figure 8: Symbolic Regression - Best 20: Program Size

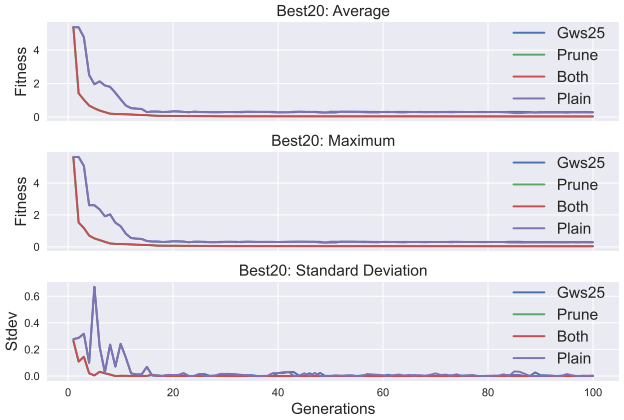

Figure 9: Symbolic Regression - Best 20: Fitness

We conclude from the results that these simplification schemes help reduce the sizes of individuals without incurring a performance penalty or loss of diversity.

4.2 Symbolic Regression

To verify the applicability of the proposed simplification algorithms, we used a classical regression problem of fitting the curve to a degree four polynomial.

$$x^4 + x^3 + x^2 + x$$

GP has already been used extensively for generating solutions for this kind of problem.

4.2.1 Experimental Setup. The operators required to generate solutions for this task are the classical mathematical operators of addition, subtraction, multiplication, and trigonometric functions. We also implemented a safe division operator to avoid division by zero. The experiments were executed using version 1.5.5 of SBCL. Each experiment was run 10 times where each run comprised 100 generations and the population size of each generation was 1,000.

4.2.2 Results. When tested on symbolic regression, most of the time pruning as an operator produced individuals that were very similar to each other across generations. It also failed to reduce the size of the individuals, as presented in figure 8. One can notice that pruning as an operator nullified the impact of generationwide simplification. Hence, the plot lines of both almost followed the trend of pruning. The standard deviation plot showed the loss in diversity whenever pruning is applied. This loss in diversity is the reason for search being stuck in local minima, thus producing individuals comprising of the same genetic makeup. Generationwide simplification, on the other hand, not only reduced the size of the individuals but also retained genetic diversity.

To make sure that there is no price of size reduction, a comparison of the fitness of the fittest 20 individuals of all generations is presented in figure 9. All the approaches are equally performing as is evident from the very close fitness lines.

To holistically study the behaviour, we compared the program sizes of all individuals across generations. Figure 10 depicts the same

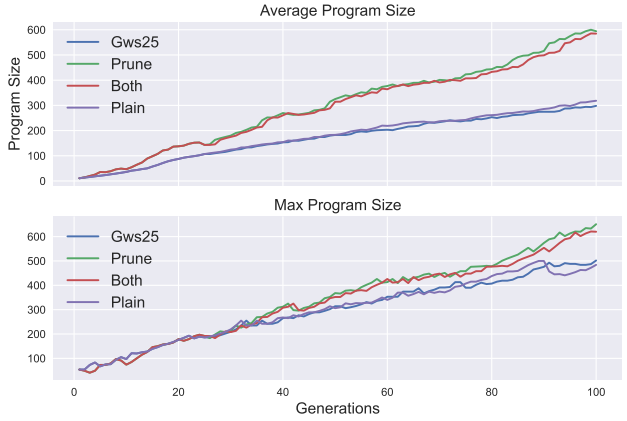

Figure 10: Symbolic Regression - Comparison of Program Sizes

trend as was observed in the case of the top 20 individuals. Hence, the hypothesis of pruning based simplification leading towards local optima is further validated. Generationwide simplification line stayed below the plain line most of the time, thus indicating the marginal reduction in program sizes.

To verify the cost paid in terms of fitness, we compare population-wide fitness. The comparison is shown in figure 11. Again, it followed the same trend that was presented in the top 20's scenario. Sometimes we obtained better fitness by applying simplification.

A comparison of diversity, both in terms of fitness and program size, is presented by calculating the standard deviation in figure12. In terms of fitness, there are no significant differences between all the schemes. However, the rate of growth of standard deviation when pruning is applied is not as fast as the generationwide simplification and plain GP. This indicates the loss of genetic diversity.

These results indicate that better solutions, with a relatively smaller size, can be obtained by applying these simplification schemes. However, pruning is prone to stick in local minima because it causes

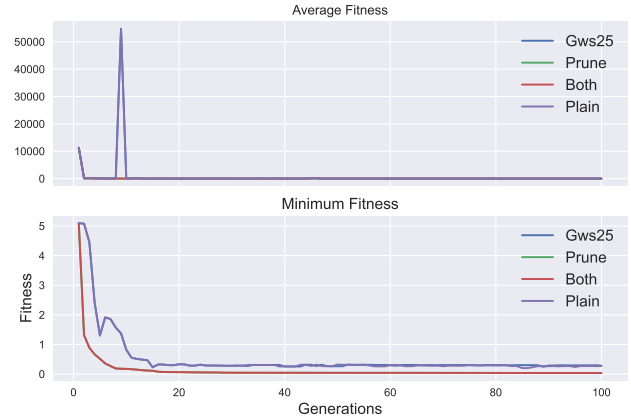

Figure 11: Symbolic Regression - Comparison of Fitness

Figure 12: Symbolic Regression - Comparison of Diversity

the loss of genetic diversity. On the other hand, generationwide simplification is robust to any such genetic loss.

5 CONCLUSION AND FUTURE WORK

White-box approaches are needed to cope with the challenge of interpretability and explainability in machine learning. GP is one such approach and has the potential to take the lead, provided it can simplify its generated models by reducing their size. This paper proposed two simplification techniques to reduce the size of GP generated models. Both of these techniques work as part of the evolutionary process, thus offering a chance to control bloat as well. An added benefit of these techniques is keeping redundancy under check. The proposed approaches are tested by implementing a well-known psychological experiment and the classical symbolic regression problem. Both these approaches performed better than the plain GP. No extra cost is incurred in the form of loss of fitness and/or loss in diversity. Both the approaches generated models of significantly smaller sizes as compared to the plain GP.

As claimed, these approaches can be extended using time as a simplification parameter rather than size. Time could be execution time or any other form of time, such as reaction time in the case of the operators used in the Posner task. The potential benefit of using time-based simplification is qualitatively comparing different operators taking part in the model generation. Another idea is to compare these simplification techniques with multi-objective fitness measures where a part of the fitness function will take care of size or time. Thus, penalising individuals with greater sizes and high time measures by assigning them low fitness.

These approaches can be made computationally more viable by storing fitness results and output vectors of the individuals. In this way, whenever required these stored results can be accessed without any need for recalculation. These stored results can also be used as a semantic measure to measure the similarity of two genetic different looking individuals. Hence, the current fitness driven simplification approaches can be extended by output result-driven measures.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. This article is part of the project "Genetically Evolving Models in Science (GEMS)" that has received funding from the European Research Council (ERC) under the grant agreement no. ERC-2018-ADG-835002.

REFERENCES

[1] Antonio Arjona, Miguel Escudero, and Carlos M. Gómez. 2016. Cue validity probability influences neural processing of targets. *Biological Psychology* 119 (2016), 171 – 183. https://doi.org/10.1016/j.biopsycho.2016.07.001

[2] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. 2020. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion* 58 (June 2020), 82–115. https://doi.org/10.1016/j.inffus.2019.12.012

[3] Tobias Blickle and Lothar Thiele. 1994. Genetic programming and redundancy.. In *Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94)*. Saarbrücken, 33–38.

[4] T. H. Chu and Q. U. Nguyen. 2017. Reducing code bloat in Genetic Programming based on subtree substituting technique. In *2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES)*. 25–30.

[5] Thi Huong Chu, Quang Uy Nguyen, and Van Loi Cao. 2018. Semantics Based Substituting Technique for Reducing Code Bloat in Genetic Programming. In *Proceedings of the Ninth International Symposium on Information and Communication Technology - SoICT 2018*. ACM Press, Danang City, Viet Nam, 77–83. https://doi.org/10.1145/3287921.3287948

[6] Francisco Fernández de Vega, Gustavo Olague, Francisco Chávez de la O, Daniel Lanza, Wolfgang Banzhaf, and Erik D. Goodman. 2020. It is Time for New Perspectives on How to Fight Bloat in GP. *CoRR* abs/2005.00603 (2020). arXiv:2005.00603 https://arxiv.org/abs/2005.00603

[7] Jean-Marc Fellous, Guillermo Sapiro, Andrew Rossi, Helen Mayberg, and Michele Ferrante. 2019. Explainable Artificial Intelligence for Neuroscience: Behavioral Neurostimulation. *Frontiers in Neuroscience* 13 (Dec. 2019), 1346. https://doi.org/10.3389/fnins.2019.01346

[8] Leonardo Augusto Ferreira, Frederico Gadelha Guimarães, and Rodrigo Silva. 2020. Applying Genetic Programming to Improve Interpretability in Machine Learning Models. *arXiv:2005.09512 [cs]* (May 2020). http://arxiv.org/abs/2005.09512 arXiv: 2005.09512.

[9] Randy Goebel, Ajay Chander, Katharina Holzinger, Freddy Lecue, Zeynep Akata, Simone Stumpf, Peter Kieseberg, and Andreas Holzinger. 2018. Explainable AI: The New 42? In *Machine Learning and Knowledge Extraction*. Vol. 11015. Springer International Publishing, Cham, 295–303. https://doi.org/10.1007/978-3-319-99740-7_21 Series Title: Lecture Notes in Computer Science.

[10] Daniel Howard and Mark A. Edwards. 2018. Explainable A.I.: The Promise of Genetic Programming Multi-run Subtree Encapsulation. In *2018 International Conference on Machine Learning and Data Engineering (iCMLDE)*. IEEE, Sydney, Australia, 158–159. https://doi.org/10.1109/iCMLDE.2018.00037

[11] David Kinzett, Mengjie Zhang, and Mark Johnston. 2008. Using Numerical Simplification to Control Bloat in Genetic Programming. In *Simulated Evolution*

and Learning. Vol. 5361. Springer Berlin Heidelberg, Berlin, Heidelberg, 493–502. https://doi.org/10.1007/978-3-540-89694-4_50 Series Title: Lecture Notes in Computer Science.

[12] John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo (Eds.). 1996. *Genetic Programming 1996: Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University*. The MIT Press. https://doi.org/10.7551/mitpress/3242.001.0001

[13] Sean Luke and Liviu Panait. 2006. A Comparison of Bloat Control Methods for Genetic Programming. *Evolutionary Computation* 14, 3 (Sept. 2006), 309–344. https://doi.org/10.1162/evco.2006.14.3.309

[14] Mori Naoki, Bob McKay, Nguyen Xuan, Essam Daryl, and Saori Takeuchi. 2009. A New Method for Simplifying Algebraic Expressions in Genetic Programming Called Equivalent Decision Simplification. In *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*. Vol. 5518. Springer Berlin Heidelberg, Berlin, Heidelberg, 171–178. https://doi.org/10.1007/978-3-642-02481-8_24 Series Title: Lecture Notes in Computer Science.

[15] Quang Uy Nguyen and Thi Huong Chu. 2020. Semantic approximation for reducing code bloat in Genetic Programming. *Swarm and Evolutionary Computation* 58 (Nov. 2020), 100729. https://doi.org/10.1016/j.swevo.2020.100729

[16] Michael I. Posner. 1980. Orienting of attention. *Quarterly Journal of Experimental Psychology* 32, 1 (1980), 3–25. https://doi.org/10.1080/00335558008248231 arXiv:https://doi.org/10.1080/00335558008248231 PMID: 7367577.

[17] Robert A. Rescorla and A. R. Wagner. 1972. A theory of Pavlovian conditioning: Variations on the effectiveness of reinforcement and non-reinforcement. In *Classical conditioning II: Current research and theory*. Appleton-Century-Crofts, New York, 64–99.

[18] Peter Rockett. 2020. Pruning of genetic programming trees using permutation tests. *Evolutionary Intelligence* (April 2020). https://doi.org/10.1007/s12065-020-00379-8

[19] Aliyu Sani Sambo, R. Muhammad Atif Azad, Yevgeniya Kovalchuk, Vivek Padmanaabhan Indramohan, and Hanifa Shah. 2020. Time Control or Size Control? Reducing Complexity and Improving Accuracy of Genetic Programming Models. In *Genetic Programming*. Vol. 12101. Springer International Publishing, Cham, 195–210. https://doi.org/10.1007/978-3-030-44094-7_13 Series Title: Lecture Notes in Computer Science.

[20] SBCL. 2019. Steel Bank Common Lisp. http://www.sbcl.org/

[21] Sara Silva, Stephen Dignum, and Leonardo Vanneschi. 2012. Operator equalisation for bloat free genetic programming and a survey of bloat control methods. *Genetic Programming and Evolvable Machines* 13, 2 (June 2012), 197–238. https://doi.org/10.1007/s10710-011-9150-5

[22] Jonas Wanner, Lukas-Valentin Herm, and Christian Janiesch. 2020. How much is the black box? The value of explainability in machine learning models.. In *ECIS 2020 Research-in-Progress Papers*. 15. https://aisel.aisnet.org/ecis2020_rip/85

[23] Phillip Wong and Mengjie Zhang. 2006. Algebraic simplification of GP programs during evolution. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06*. ACM Press, Seattle, Washington, USA, 927. https://doi.org/10.1145/1143997.1144156

[24] Phillip Wong and Mengjie Zhang. 2007. Effects of program simplification on simple building blocks in Genetic Programming. In *2007 IEEE Congress on Evolutionary Computation*. IEEE, Singapore, 1570–1577. https://doi.org/10.1109/CEC.2007.4424660